# Assignment 2: Report

Name:
ID:

Based on your solution and understanding of the assignment and lecture slides, address the questions below:

1. Identify opportunities for parallelism in the sequential version of the code by stating which loops can be parallelized using OpenMP. Consider all loops of the sequential code and explain why these loops can be parallelized, and how did you proceed to parallelize them with OpenMP.

In the sequential code, initialization & boundary loops, the loop for computation of new values and the loop for copying interior points from matrix W to U, can be parallelized because each iteration operates on a different matrix element and iterations are independent. They were parallelized by using the following OpenMP clauses:

*#pragma omp parallel for default(none) shared(U, W) private(i, j) firstprivate(M, N)*

*#pragma omp parallel for default(none) shared(U, W) private(j) firstprivate(M, N)*

*#pragma omp parallel for default(none) shared(U, W) private(i, j) reduction(+:diffnorm) firstprivate(M, N)*

*#pragma omp parallel for default(none) shared(U, W) private(i, j) firstprivate(M, N)*

In each case, the **#pragma omp parallel for** directive is used to indicate that the loop should be executed in parallel. The **default(none)** clause is added to enforce explicit data sharing, and **private** clauses are used to declare private variables for each thread. Additionally, **reduction** clause is used where necessary to handle reduction operations.

2. Was there any loop in your OpenMP code that you could not parallelize? Explain why you could not parallelize it via data dependency analysis.

The inner loop for computing new values involves updating elements of the **W** matrix based on neighboring elements of the **U** matrix. This dependency on neighboring elements could introduce data dependencies that might interfere with parallelization.

The value of **W[i][j]** depends on the values of **U[i][j + 1]**, **U[i][j - 1]**, **U[i + 1][j]**, and **U[i-1][j]**.

Since each iteration updates elements in the **W** matrix based on the current values of neighbouring elements in the **U** matrix, there are potential data dependencies between iterations.

3. In your OpenMP code, you were supposed to explicitly define the data-scope of all variables that are used within a parallel region. Justify your data-scoping choices for each variable within the extent of a parallel region.

Shared variables are matrices 'U' and 'W'. All threads need access to these matrices to read and update the values of elements during the computation.

Private variables are loop indices 'I' and 'j'. They are declared as private because each thread must have their own copy of these variables to iterate independently over the assigned portion of the work. They should be private to avoid race conditions and ensure each thread operated on its subset of matrix elements without interfering with other threads.

Firstprivate variables are the loop dimensions 'M' and 'N'. They are declared as firstprivate because each thread should work with their own private copy of these values to ensure that threads operate on the specified range without interference, preventing potential race conditions.

Lastly, 'diffnorm' was declared as reduction variable because it is updated independently by each thread within the loop, and its final value needs to be computed by combining these private copies after the loop.

4. Was there any variable in your parallel code that you had to protect with OpenMP synchronization constructs? If yes, identify the variable, and explain how you protected the accesses to this variable.

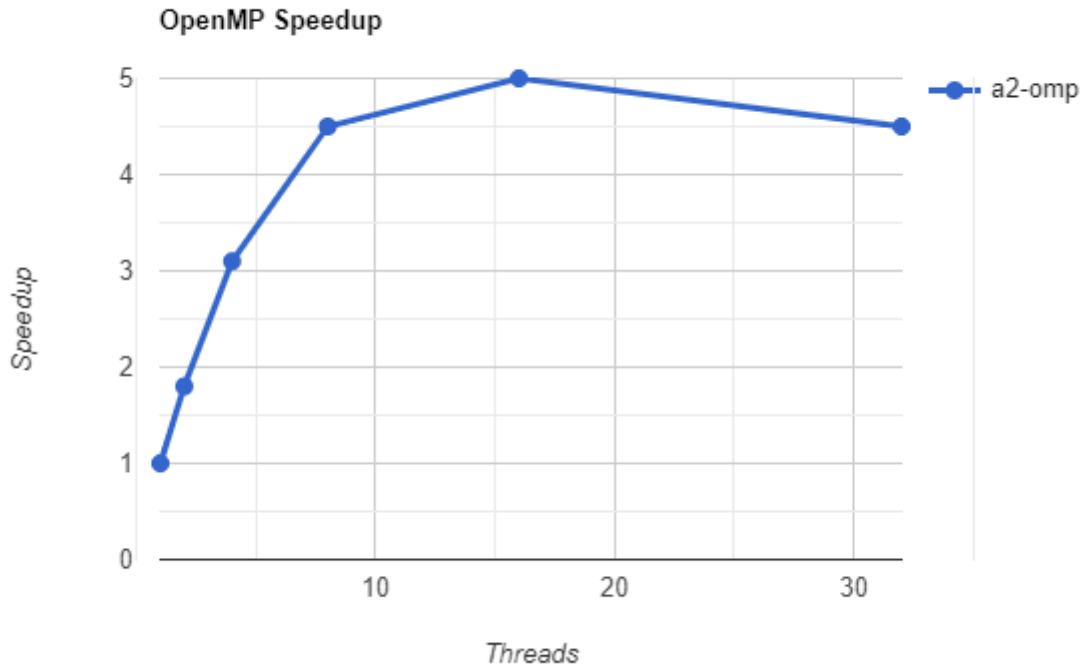
In my code, I have not used any explicit synchronization constructs such as critical sections, atomic operations or locks.

5. Consider the two `for`-loops inside the `while-do` loop of `a2.cpp`. Assuming you parallelized both of these `for`-loops with OpenMP, discuss whether the `nowait` clause could be applied to the first `for`-loop.

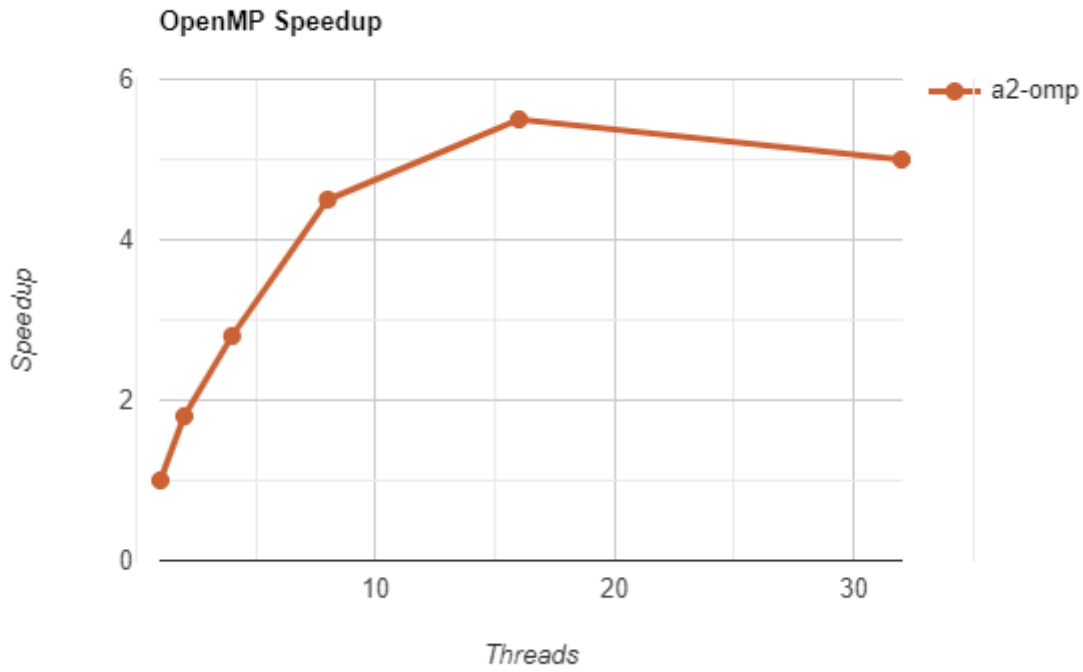
Applying **nowait** to the first for-loop would mean that the threads are not synchronized after the loop, and the subsequent part of the computation might access inconsistent data. This would lead to incorrect results. The subsequent part of the loop depends on the values computed in the parallel region, so it's crucial to ensure that the parallel region completes before proceeding to the next part of the loop.

6. Include a speedup plot of your OpenMP solutions on ALMA (for 1, 2, 4, 8, 16 and 32 threads). Remember: Use the horizontal axis to represent the number of cores, while the vertical is used to represent the speedup (not execution time).

**OMP_NUM_THREADS=… srun --nodes=1 ./a2-omp --m 2688 --n 4096 --epsilon 0.01 – max-iterations 1000**




**OMP_NUM_THREADS=… srun --nodes=1 ./a2-omp --m 2688 --n 4096 --epsilon 0.01 – max-iterations 2000**



**OMP_NUM_THREADS=… srun --nodes=1 ./a2-omp --m 1152 --n 1152 --epsilon 0.01 –max-iterations 1000**